Climate Data Analysis Tools

Version 3

May, 2001

Program for Climate Model Diagnosis and Intercomparison (PCMDI) Lawrence Livermore National Laboratory, Livermore California 94550

Legal Notice

Table of Contents

CHAPTER 1

Getting Started With CDAT

CDAT is an open-source, Python-based set of tools for scientific data analysis and graphics.

1.1 Introduction

1.1.1 A New Paradigm

The software you are about to use, CDAT (Climate Data Analysis Tools), may be unlike any software you may have used before. Rather than a monolithic interface with a fixed number of predetermined commands, CDAT is a set of components that can be programmed under the direction of the user. The set of components can be extended with additional algorithms and compiled code written by the user or written by other members of the community. This kind of extension does not require the cooperation or consent of the CDAT authors or even the recompilation of any of CDAT itself.

CDAT is based on the popular scripting language Python (http://www.python.org). Python is easy to learn, and most importantly for our purposes, easy to extend with your own compiled code and additional modules written in Python itself. In fact, adding your own C, C++, or Fortran can be done nearly automatically using tools that have been written for the purpose. (See “User-contributed packages” on page17.)

Some of the key components supplied by PCMDI include:

- cdms, a package that enables access to many different data file formats;
- cdutil, a package containing utility functions for climatology and averaging;
- genutil, a package containing statistics functions with special features for climate data;
- vcs, a two-dimensional graphics package.

1.1.2 The CDAT Project at SourceForge

CDAT is hosted at SourceForge, a free service provided by VA Linux, Inc. to the Open Source Community. Using Sourceforge enables us to have many services for users:

- A release facility, where users can download binary and source releases and see release notes.
- A bug-tracking facility, where users can submit bugs and track their status, and receive mail when they are fixed.
- A mailing list for discussion of CDAT.

You can use these facilities without registering at SourceForge, but registration, which is quick, easy, and free, will enable you to participate in the fullest possible way. In particular, it is very helpful to us if you are registered when you submit a bug report.

The **CDAT Home Page** is cdat.sourceforge.net. That page documentation and links to related sites. One link is to the **CDAT Project Page**, sourceforge.net/projects/c/cd/cdat. (No, we aren't stuttering; there are so many projects on SourceForge that they had to organize their web site this way). The Project Page contains the bug-tracking, mail list, and download facilities.

1.2 Setting up CDAT

CDAT is available in binary form for a number of platforms. See the CDAT Project Page for the available downloads. It is also available in source form. With the source form, you pick an installation directory. With the binary form, you simply undo the tarball where desired. The top level of the tarball will have a name like cdat-3.0.0.

Either form of CDAT can be installed anywhere on your computer. Wherever you install it, it is important to do two things:

1. Put the 'bin' directory of the installation in your Unix search path so that executing

   where python

   prints the path to the python in your CDAT installation. For example, if you have untarred a binary into /usr/local/, you want /usr/local/cdat-3.0.0/bin in your path, and the above command should print "/usr/local/cdat-3.0.0/bin/python". Depending on your OS, you may need to issue a "rehash" command after doing this. Typically, you are going to want to fix this permanently in your startup files.
2. Set the environment variable LD_LIBRARY_PATH to include the directories where Netcdf, tcl, and tk are installed. If you do not have these, you will need to obtain them and install them first. Usually they are placed in /usr/local.

To build from source, untar the distribution into any directory of your own and proceed to follow the instructions in the README file at the top level.

A first test

Execute "python" and a prompt >>> should appear. Enter the commands:

```
>>> import cdms
>>> import vcs
```

If you get back another prompt after each command, CDAT is installed correctly. You can leave python by entering “Control-D”.

Troubleshooting

If you get an error message about not being able to load a library such as libnetcdf.so, it means you didn’t set LD_LIBRARY_PATH.

If python says it can’t find modules such as cdms or vcs, it means you don’t have our python at the front of your path. Linux, for example, comes with a python already installed so you might be running the wrong one. We have to give you one of our own to make sure your Python version matches the one with which we built all the CDAT components.

1.3 Learning Python

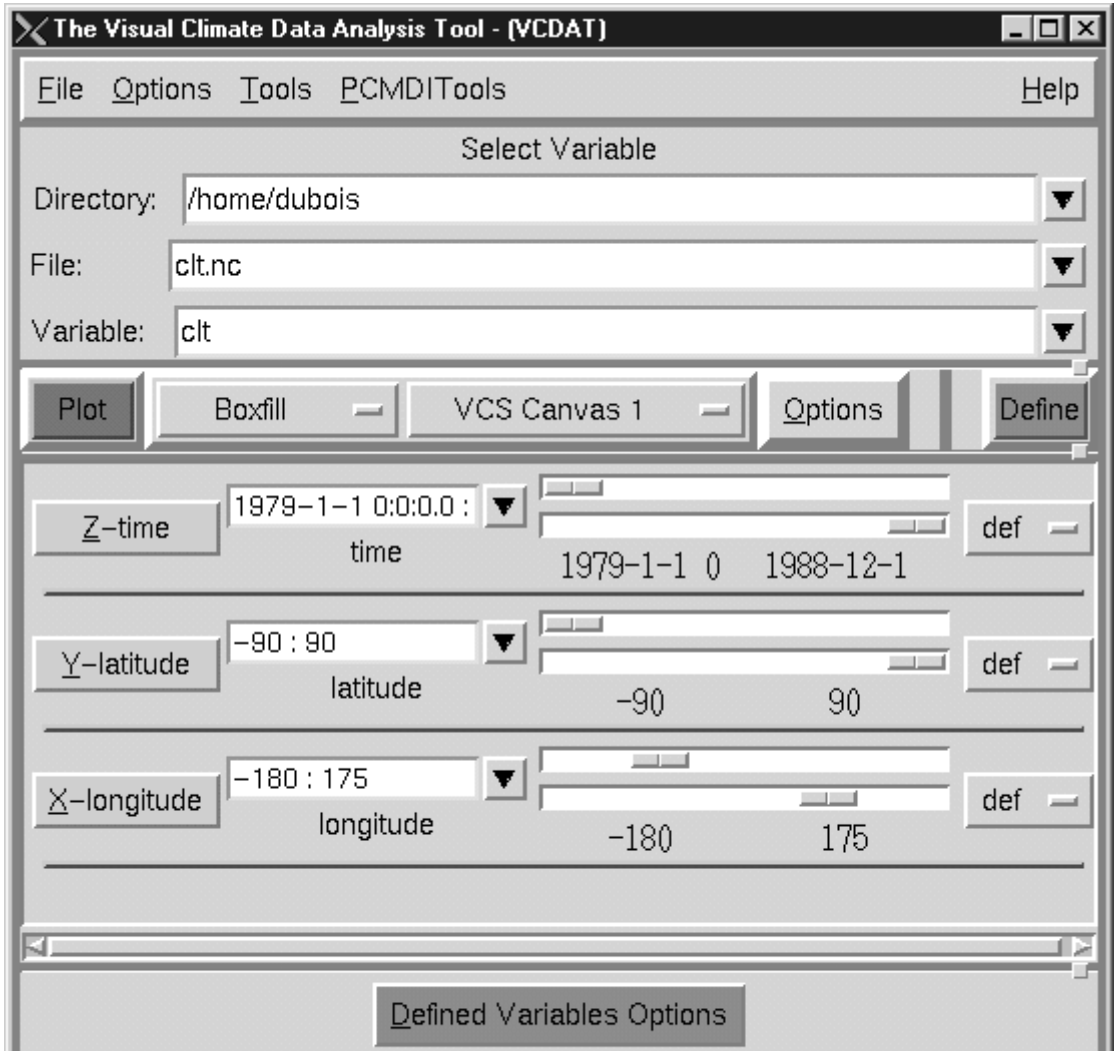
Python is documented down to the last detail at python.org and the SourceForge website python.sourceforge.net. There are now many books about it. We can recommend the free tutorial available at python.org in the documentation section. The book “Learning Python” from O’Reilly Press is another excellent resource.

CDAT makes heavy use of Numerical Python, a fast array facility for Python. The documentation for Numerical Python is at numpy.sourceforge.net.

Although CDAT itself is not yet available on Windows or MacIntosh, Python is. Python is even available for the Palm Pilot. You can download a Windows installer and even install Numerical Python on Windows (see numpy.sourceforge.net).

You may also wish to run Python in a nicer environment than a shell. The IDLE environment can be started by entering “idle”. Documentation for IDLE is available via its “help” button. Many people run Python from Emacs. Information about how to configure your editor for writing Python code is available on the Python site.

1.4 The VCDAT GUI



There is no easier way to use CDAT than by starting up the Visual CDAT application, vcdat. If you have Python installed correctly as described above, then executing “vcdat” should be all you need to do.

We think most users will be able to do a lot with VCDAT without reading any instructions. Give it a try, and if you have questions consult the VCDAT documentation on our home page.

For some users, this powerful tool may suffice for most of their needs. It even contains facilities within it to help you learn how to write the scripts that can do what you want to do, so that you can do those things without VCDAT.

CHAPTER 2

Basic tutorials

CDAT comes with a suite of tutorials to help you learn how to use it. You can get these tutorials and sample data from our website.

2.1 How to get the tutorials and data

The tutorials are designed to introduce you to the most common operations for climate data analysis. They are available as a Python script from our download area under "Tutorials". If you want to try executing these examples, do these steps:

1. From the Tutorials area of the download facility at cdat.sf.net, download the data files tarball and unpack it.
2. Download the tutorial files. In each tutorial you will need to edit the line(s) near the top that sets the location of the data files to match the location where the data will be on your system.

We will now describe the tutorials.

2.2 getting_started_tutorial.py

This tutorial is the first one to study. The tutorial consists of three parts:

Example 0: the basics

- How do I open a file?
- How do I see what variables are in the file?
- How do I read a variable?
- How can I get the metadata for the variable?
- How do I get the shape (i.e length of each dimension) of the variable?
- How do I extract a variable and rewrite it to another file?
- How do I extract data at a specific latitude/longitude from the file?
- I do not want to retain the latitude and longitude axes since they are single points. How do I do that?
- How do I extract a subset (or a region) from the file?
- How can I define a region of interest and reuse it without a lot of typing?
- How do I see what the latitude and longitude values extracted are?
- What if I need say the 2nd axis whose name I do not know?
- How do I get the exact region with the precise bounds and not any spillover into adjoining grid cells?
- How do I get the area averaged NINO3 values?
- How do I see what the start time in the data set is?
- How do I extract the data based on time axis values since I know the first time point in the data?
- How do I extract a time slice with specific time start and end ?

Example 1: Dealing with data from other sources

- Reading ASCII data like that written by Fortran
- "Masking" or setting a certain value as "Missing data"
- How to create an Axis
- How to set the name of the axis

- How to set the axis units.
- How to create Uniform Latitude and Longitude axes.
- How to get the Bounds.
- How to create a "variable" with all the metadata from an array.
- What are the options for createVariable?
- How to check the shape of the variable
- How to check the metadata or decorations to the variable and its axes.
- How to plot a variable using VCS
- How to write a variable out in a netcdf file.
- Averaging with weights over specified dimensions.
- Specifying weights.
- Generating weights.

Example 2. Masking out data using a land fraction data file.

This example opens a temperature data file and a corresponding land fraction data file. Use the land fraction data to select land/ ocean areas from the temperature data. After masking out data use the averaging routines to compute the avea averages.

2.3 Statistics

statistics_tutorial.py illustrates the use of the tools for calculating statistics such as covariances on climate data.

2.4 Dealing with time

Dealing with time is one of the hardest parts of dealing with climate data files. times_tutorial.py illustrates how to use the facilities in CDAT that make dealing with time less painless. Examples are given of:

- Compute the climatological DJF
- Calculate departures from that
- Compute the departures from the 1979-1988 period
- Using the predefined seasons:
  JAN, FEB, MAR, APR, ..., DEC
  DJF, MAM, JJA, SON
  YEAR -- returns annual means
  ANNUALCYCLE -- returns monthly means for each month of the year
  SEASONALCYCLE -- returns seasonal means for the 4 pre-defined season
- Compute the annual mean for each year
- Compute the global average for each month

2.5 Plotting with xmgrace

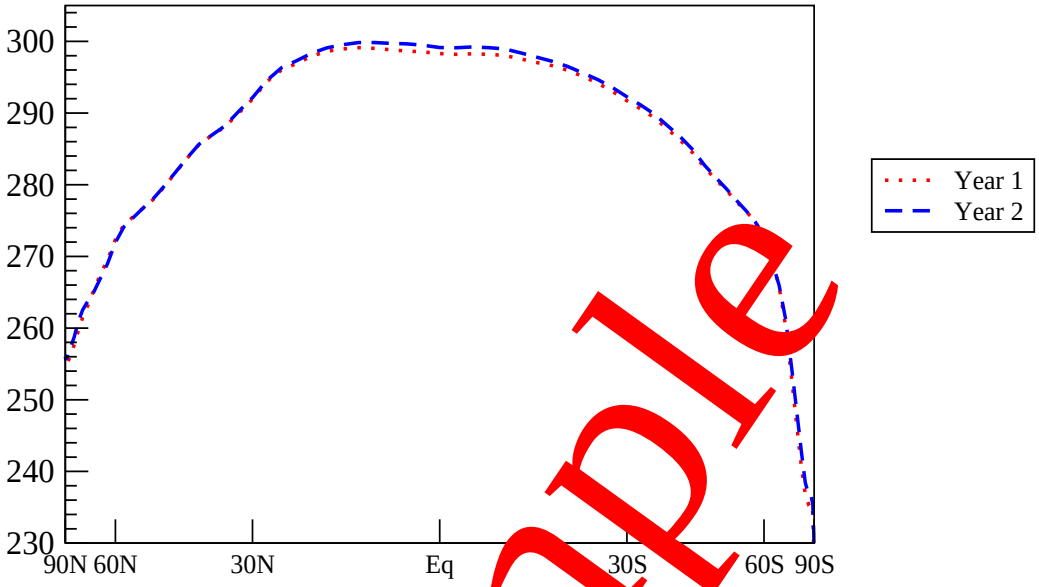
Nothing emphases the fact that CDAT is a collection of tools that can be extended by the user better than the xmgrace module. This module provides an interface to the popular xmgrace utility (which you must have installed yourself separately). One of our users, Charles Doutriaux, who loved xmgrace built this interface. The xmgrace tutorial will teach you how to use it.

The plot shown on the next page was produced with the xmgrace module tutorial (but it had to be scaled down to fit the page).

A spreadsheet xmgrace.xls is available via the website. This spreadsheet contains detailed information needed by xmgrace users.

NCEP Reanalysis, Surface air temperature

First and Last year



Difference

CHAPTER 1

Creating New Packages

1.1 Adding your science

One of CDAT's strengths is that it is an open system. You can add your own software written in C, Python, or Fortran. The easiest way to learn to do this is to copy our examples. Get the CDAT source distribution and look for sub-directory 'contrib' in the top-level directory. The README file in contrib explains what to do.

There are tools that may be useful to you.

- The SWIG utility (Simplified Wrapper and Interface Generator, www.swig.org) can wrap C and C++ routines.
- Pyfort (pyfortran.sourceforge.net) connects Fortran routines to Python.

Depending on your needs, you may wish to use a layer of Python along with the automatically created interface, in order to make a nicer interface or to use the Fortran or C simply as computational engines. An example of this is the EOF package described below: it uses a Fortran linear algebra routine to enhance performance, but the "science" is in Python.

If you follow the protocols in 'contrib' then your package can be added to the PCMDI distribution as well. Just send it to us and be sure to include a README that explains:

- How to use the package
- Contact information about the author.

You may also be able to generate useful documentation by executing the routines happydoc or pydoc. happydoc works only on Python code; pydoc works on the installed modules. Both routines print help packages if executed with the argument, '--help', and both are already installed in your cdat 'bin' directory.

CHAPTER 2

User-contributed packages

If you have the source distribution, use the README files in the subdirectories of the contrib directory for full documentation. Try running pydoc -w with the name of the package as an argument to create a web page showing the package's interface.

Due to the very nature of the user-contributed packages, the following list of available packages and their exact capabilities may be incomplete or inaccurate. PCMDI does not maintain all of these packages, and is not responsible for fixing bugs them -- please contact the author.

2.1 Package: asciidata

Author: Paul Dubois (dubois1@llnl.gov)

Summary: Package asciidat can be used to read text files written by such programs as spreadsheets, in which data has been written as comma, tab, or space-separated numbers with a header line that names the fields. Using the functions in asciidata, you can convert these columns into Numerical arrays, with control over the type/precision of these arrays.

Example

```
import asciidata
time, pressure = asciidata.comma_separated('myfile.csv')
```

Documentation: pydoc -w asciidata

2.2 Package: binaryio

Author: Paul F. Dubois

Summary: Read and write Fortran unformatted i/o files. These are the files that you read and write in Fortran with statements like read(7) or write(7).

Such files have an unspecified format and are platform and compiler dependent. They are NOT portable. Contrary to popular opionion, they are NOT standard. The standard only specifies their existance and behavior, not the details of their implementation, and since there is no one obvious implementation, Fortran compilers *do* vary. We suggest writing netcdf files instead, using the facilities in cdms.

Documentation: pydoc -w binaryio

2.2.1 Usage summary:

```
binaryio: Fortran unformatted io
   Uses Fortran wrapper module "binout"
Usage:
   from binaryio import *
   iunit = bincreate('filename')
   binwrite(iunit, some_array)  (up to 4 dimensions, or scalars)
   binclose(iunit)
   iunit = binopen('filename')
   y = binread(iunit, n, ...)   (1-4 dimensions)
   binclose(iunit)
```

Note that reads and writes must be paired exactly. Errors will cause a Fortrastop that cannot be recovered from. You must know (or have written earliein the file) the sizes of each array.

All data is stored as 32-bit floats.

2.3 Package: eof

Author: Paul Dubois (dubois1@llnl.gov) based on work by Ken Sperber and Ben Santer.

Summary: Package eof calculates Explicit Orthonormal Functions of either one variable or two variables jointly. Having selected some data, the key call is to create an instance of eof.Eof giving one or two arguments. In this example, a portion of the variable 'u' is analyzed. After the result is returned, it is an object with attributes containing such things as the principal components and the percent of variance explained. Optional arguments are available for controlling the subtraction of the mean from the data, the weighting by latitude, and the number of components to compute.

This routine is computationally efficient, solving the problem in either the normal space or the dual space in order to minimize computations. Nonetheless, it is possible that this routine will require substantial time and space if used on a large amount of data. This cost is determined by the smaller of the number of time points and the total number of space points.

Documentation: pydoc -w eof.Eof

2.3.1 Example

```
import cdms, vcs
from eof import Eof

f=cdms.open('/home/dubois/clt.nc')
u = f('u', latitude=(-20,40), longitude=(60, 120))
result = Eof(u)
principal_components = result.principal_components
print "Percent explained", result.percent_explained
x=vcs.init()
vcs.pauser.pause(3)
print len(principal_components)
for y in principal_components:
   x.isofill(y)
```

```
   x.clear()
u1 = v.subRegion(latitude=(amr[0], amr[1], 'cc'),
            longitude=(amr[2],amr[3],'cc'), order='xyt')
result2 = Eof(u, number_of_components=4,
mean_choice=12)
print "Percent explained", result.percent_explained
```

2.4 Package: lmoments

Author: Michael Werner based on L-moments library by J. R. M. Hosking

Summary: This package is an interface to a Fortran library. The calling sequence from Python differs from the Fortran convention. In general, output and temporary arguments are not supplied in making the Python call, and output arguments are returned as values of the function.

Documentation: pydoc -w lmoments to see list of functions. pydoc -w lmoments.pelexp, or other function name, for the particular. See also documentation for Pyfort at pyfortran.sourceforge.net for further details on argument conventions. If built from source, a file flmoments.txt appears which gives the Python calling sequences.

2.5 Package: regridpack

Author: Clyde Dease

Summary: Interface to regridpack

Documentation: This package contains a Python interface to the subroutine library regridpack.

pydoc -w adamsregrid Documentation online at cdat.sourceforge.net. See also documentation for Pyfort at pyfortran.sourceforge.net for further details on argument conventions.

2.6 Package: sphere (spherepack)

Author: Clyde Dease

Summary: Interface to Spherepack

Documentation: This package contains a Python interface to the subroutine library Spherepack.

pydoc -w sphere to see list of functions. Documentation online at cdat.sourceforge.net. See also documentation for Pyfort at pyfortran.sourceforge.net for further details on argument conventions.

2.7 trends

Author: Pyfort wrapping by Paul Dubois of a routine by Ben Santer

Summary: Computes variance estimate taking auto-correlation into account.

Documentation:

```
import reg_arl from trends
rneff, result, res, cxx, rxx = reg_arl (lag, x, y)
    integer lag         Max lag for autocorrelations.
    real x(n1)          Independent variable
    real y(n1)          Dependent variable
    real, intent(out):: rneff         !Effective sample size
    real, intent(out):: result(31)    !Array of linear regression
results
    real, intent(out):: res(n1)     !Residuals from linear regres-
sion
    real, intent(out):: cxx(1 + lag)  !Autocovariance function
    real, intent(out):: rxx(1 + lag)  !Autocorrelation function
```

2.8 Package: ort

Author: Curt Covey

Summary: Read data from an OORT file.

Documentation: Module ort contains one Fortran function, read1f:

2.8.1 Calling sequence:

```
import ort
lon, lat, data, nr = ort.read1f(filename, maxsta, nvarbs,
                                nlevels)
```

Input:

```
     character*(*) filename    ! name of the file to be read
! max number of stations (soundings) possible
     integer maxsta
! number of variables and P-levels in each sounding
     integer nvarbs, nlevels
```

Output:

```
 ! longitudes / latitudes of the stations
     real, intent(out)::  lon(maxsta), lat(maxsta)
! sounding data
     real , intent(out):: data(nvarbs, nlevels, maxsta)
 ! actual number of stations with data
     integer , intent(out):: nr
```